Distributed Systems 2
Epidemic Dissemination

Alberto Montresor

Università di Trento

2021/09/26

Table of contents

Data dissemination

Problem

- Application-level broadcast/multicast is an important building block to create modern distributed applications
  - Video streaming
  - RSS feeds
- Want efficiency, robustness, speed when scaling
  - Flooding (reliable broadcast) is robust, but inefficient ($O(n^2)$)
  - Tree distribution is efficient ($O(n)$), but fragile
  - Gossip is both efficient ($O(n \log n)$) and robust, but has relative high latency
- Scalability:
  - Total number of messages sent by all nodes
  - Number of messages sent by each of the nodes

Trade-offs

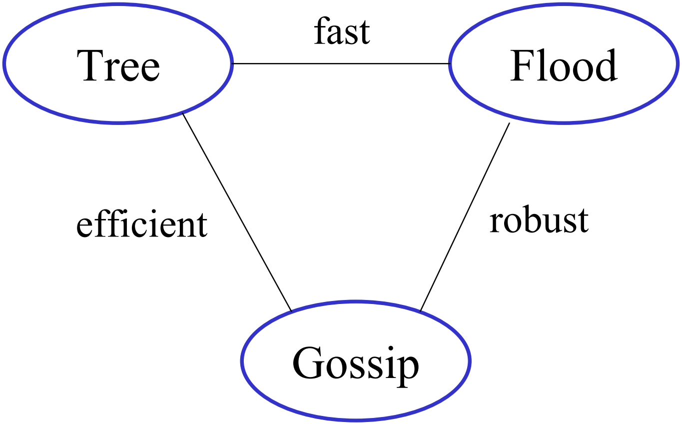


Figure: Courtesy: Robbert van Renesse

Introduction

Solution

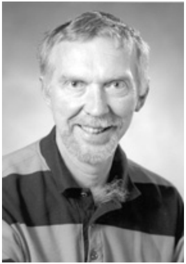
- Reverting to a probabilistic approach based on epidemics / gossip
- Nodes infect each other trough messages
- Total number of messages is less than $O(n^2)$
- No node is overloaded

But:

- No deterministic guarantee on reliability
- Only probabilistic ones

History of the Epidemic/Gossip Paradigm

- First defined by Alan Demers et al. (1987)
- Protocols based on epidemics predate Demers' paper (e.g. NNTP)
- '90s: gossip applied to the information dissemination problem
- '00s: gossip beyond dissemination
- 2006: Workshop on the future of gossip (Leiden, the Netherlands)



Reality check (1987)

XEROX Clearinghouse Servers

- Database replicated at thousands of nodes
- Heterogeneous, unreliable network
- Independent updates to single elements of the DB are injected at multiple nodes
- Updates must propagate to all nodes or be supplanted by later updates of the same element
- Replicas become consistent after no more new updates
- Assuming a reasonable update rate, most information at any given replica is "current"

Reality check (today)

- Amazon reportedly adopted a gossip protocol to quickly spread information throughout the S3 system
- Amazon's Dynamo uses a gossip-based failure detection service
- Open-source key-value stores like Riak and Cassandra are based on gossip protocols
- The basic information exchange in BitTorrent is based on gossip

Basic assumptions

- System is asynchronous
  - No bounds on messages and process execution delays
- Processes fail by crashing
  - stop executing actions after the crash
  - We do not consider Byzantine failures
- Communication is subject to benign failures
  - Message omission
  - No message corruption, creation, duplication

Data model

- We consider a database that is replicated at a set of n nodes $P = \{p_1, \ldots, p_n\}$
- The copy of the database at node p_i can be represented by a time-varying partial function:
$$value : K \to V \cdot T$$
where:
  - K is the set of keys
  - V is the set of values
  - T is the set of timestamps
- The update operation is formalized as: $value(k) \leftarrow (v, \mathsf{now}())$ where $\mathsf{now}()$ returns a globally unique timestamp

Database Specification

The goal of the update distribution process is to drive the system towards consistency:

Definition: Eventual consistency

If no new updates are injected after some time t, eventually all correct nodes will obtain the same copy of the database:

$$\forall p_i, p_j \in P, \forall k \in K : value_i(k) = value_j(k)$$

where $value_i$ is the copy of the database at node p_i.

Probabilistic Broadcast Specification

An alternative view

PB1 – Probabilistic validity

There is a given probability such that for any two correct nodes p and q, every message PB-broadcast by p is eventually PB-delivered by q

PB2 - Integrity

Every message is PB-delivered by a node at most once, and only if it was previously PB-broadcast

Best-Effort Broadcast vs Probabilistic Broadcast

Similarities:

- No agreement property

Differences:

- Probability in BEB is “hidden” in process life-cycle
- Probability in PB is explicit

Some simplifying assumptions

- Every node knows P (the communication network is a full graph)
- Communication costs between nodes are homogeneous
- We assume there is a single entry in the database
  - We simplify notation
    - $value(k)$ can be simply written as $value$
    - The timestamp is denoted $value.time$
  - Managing multiple keys is more complex than adding **foreach** $k \in K$

To be relaxed later...

Models of epidemics

Epidemiology

Epidemiology studies the spread of a disease or infection in terms of populations of infected/uninfected individuals and their rates of change

Why?

- To understand if an epidemic will turn into a pandemic
- To adopt countermeasures to reduce the rate of infection
  - Inoculation
  - Isolation

SIR Model

SIR Model – Kermack and McKendrick, 1927

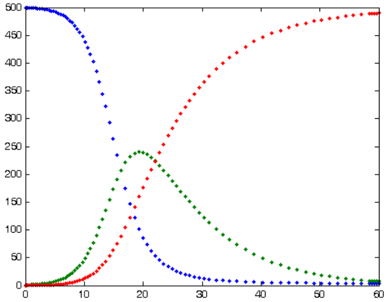
An individual p can be:

- Susceptible: if p is not yet infected by the disease
- Infective: if p is infected and capable to spread the disease
- Removed: if p has been infected and has recovered from the disease

SIR Model

How does it work?

- Initially, a single individual is infective
- Individuals get in touch with each other, spreading the disease
- Susceptible individuals are turned into infective ones
- Eventually, infective individuals will become removed



http://en.wikipedia.org/wiki/File:Sirsys-p9.png

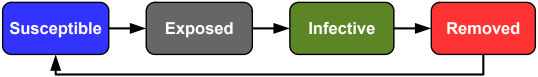
SIR is not the only model...

SIRS Model

The SIR model plus temporary immunity, so recovered nodes may become susceptible again.

SEIRS Model

The SEIRS model takes into consideration the exposed or latent period of the disease



From Epidemiology to Distributed Systems

The idea

- Diseases spread quickly and robustly
- Our goal is to spread an update as fast and as reliable as possible
- Can we apply these ideas to distributed systems?

SIR Model for Database replication

- Susceptible: if p has not yet received an update
- Infective: if p holds an update and is willing to share it
- Removed: if p has the update but is no longer willing to share it

Note: Rumor spreading, or gossiping, is based on the same principles

Table of contents

Algorithm – Summary

- Best effort
- Anti-entropy (simple epidemics)
  - Push
  - Pull
  - Push-pull
- Rumor mongering (complex epidemics)
  - Push
  - Pull
  - Push-pull
- Probabilistic broadcast

Best-effort

Best-effort (direct mail) algorithm

- Notify **all** other nodes of an update as soon as it occurs.
- When receiving an update, check if it is "news"

Direct mail protocol executed by process p_i:

upon $value \leftarrow (v, \mathsf{now}())$ **do**
  foreach $p_j \in P$ **do**
    send $\langle \textsc{update}, value \rangle$ **to** p_j

upon receive $\langle \textsc{update}, (v, t) \rangle$ **do**
  if $value.time < t$ **then**
    $value \leftarrow (v, t)$

Not randomized nor epidemic algorithm: just the simplest

Anti-entropy

Anti-entropy: Algorithm

- Every node regularly chooses another node at random and exchanges database contents, resolving differences.
- Nodes are either
  - susceptible – they don't know the update
  - infective – they know the update

Anti-entropy protocol executed by process p_i:

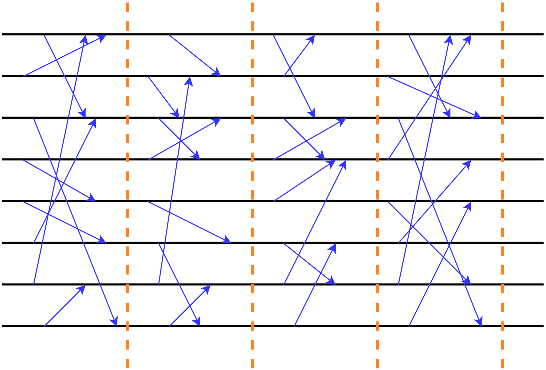
repeat every Δ time units
  $p_j \leftarrow \mathsf{random}(P)$ % Select a random neighbor
  { exchange messages to resolve differences }

Anti-entropy: graphical representation

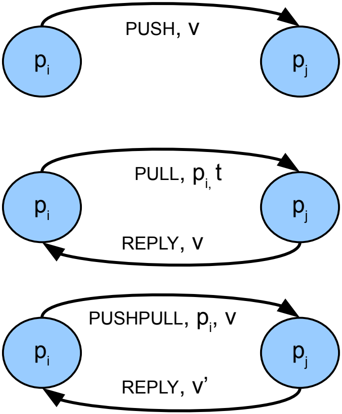
Rounds

During a round of length Δ

- every node has the possibility of contacting one random node
- can be contacted by several nodes



Resolving differences – Summary



Resolving differences – Push

Anti-entropy, Push protocol executed by process p_i:

upon timeout do
- $q \leftarrow \mathsf{random}(P)$
- **send** $\langle \text{PUSH}, \mathit{value} \rangle$ **to** q
- **set timeout** Δ

upon receive $\langle \text{PUSH}, v \rangle$ **do**
- **if** $\mathit{value}.\mathit{time} < v.\mathit{time}$ **then**
  - $\mathit{value} \leftarrow v$

Resolving differences – Pull

Anti-entropy, Pull protocol executed by process p_i:

upon timeout do
- $q \leftarrow \mathsf{random}(P)$
- **send** $\langle \text{PULL}, p, \mathit{value.time} \rangle$ **to** q
- **set timeout** Δ

upon receive $\langle \text{PULL}, q, t \rangle$ **do**
- **if** $\mathit{value.time} > t$ **then**
  - **send** $\langle \text{REPLY}, \mathit{value} \rangle$ **to** q

upon receive $\langle \text{REPLY}, v \rangle$ **do**
- **if** $\mathit{value.time} < v.\mathit{time}$ **then**
  - $\mathit{value} \leftarrow v$

Resolving differences – Push-Pull

Anti-entropy, Push-Pull protocol executed by process p_i:

upon timeout do
- $q \leftarrow \mathsf{random}(P)$
- **send** $\langle \text{PUSHPULL}, p, \mathit{value} \rangle$ **to** q
- **set timeout** Δ

upon receive $\langle \text{PUSHPULL}, q, v \rangle$ **do**
- **if** $\mathit{value}.\mathit{time} < v.\mathit{time}$ **then**
  - $\mathit{value} \leftarrow v$
- **else if** $\mathit{value}.\mathit{time} > v.\mathit{time}$ **then**
  - **send** $\langle \text{REPLY}, \mathit{value} \rangle$ **to** q

upon receive $\langle \text{REPLY}, v \rangle$ **do**
- **if** $\mathit{value}.\mathit{time} < v.\mathit{time}$ **then**
  - $\mathit{value} \leftarrow v$

Analytical results

Compartmental model analysis

We want to evaluate convergence of the protocol based on the size of the populations of susceptible and infected nodes (compartments)

- s_t: the probability of a node being susceptible after t anti-entropy rounds
- $i_t = 1 - s_t$: the probability of a node being infective after t anti-entropy rounds

Probability at round $t + 1$

- Initial condition: $s_0 =$
- Pull: $E[s_{t+1}] =$
- Push: $E[s_{t+1}] =$

Analytical results

Compartmental model analysis

We want to evaluate convergence of the protocol based on the size of the populations of susceptible and infected nodes (compartments)

- s_t: the probability of a node being susceptible after t anti-entropy rounds
- $i_t = 1 - s_t$: the probability of a node being infective after t anti-entropy rounds

Probability at round $t + 1$

- Initial condition: $s_0 = \frac{n-1}{n}$
- Pull: $E[s_{t+1}] =$
- Push: $E[s_{t+1}] =$

Analytical results

Compartmental model analysis

We want to evaluate convergence of the protocol based on the size of the populations of susceptible and infected nodes (compartments)

- s_t: the probability of a node being susceptible after t anti-entropy rounds
- $i_t = 1 - s_t$: the probability of a node being infective after t anti-entropy rounds

Probability at round $t + 1$

- Initial condition: $s_0 = \frac{n-1}{n}$
- Pull: $E[s_{t+1}] = s_t^2$
- Push: $E[s_{t+1}] =$

Analytical results

Compartmental model analysis

We want to evaluate convergence of the protocol based on the size of the populations of susceptible and infected nodes (compartments)

- s_t: the probability of a node being susceptible after t anti-entropy rounds
- $i_t = 1 - s_t$: the probability of a node being infective after t anti-entropy rounds

Probability at round $t + 1$

- Initial condition: $s_0 = \frac{n-1}{n}$
- Pull: $E[s_{t+1}] = s_t^2$
- Push: $E[s_{t+1}] = s_t(1 - \frac{1}{n})^{(1-s_t)n}$

Analytical results

Compartmental model analysis

We want to evaluate convergence of the protocol based on the size of the populations of susceptible and infected nodes (compartments)

- s_t: the probability of a node being susceptible after t anti-entropy rounds
- $i_t = 1 - s_t$: the probability of a node being infective after t anti-entropy rounds

Probability at round $t + 1$

- Initial condition: $s_0 = \frac{n-1}{n}$
- Pull: $E[s_{t+1}] = s_t^2$
- Push: $E[s_{t+1}] = s_t(1 - \frac{1}{n})^{(1-s_t)n} \approx s_t e^{-(1-s_t)}$

Analytical results

Termination time

Let $T(n)$ be the expected round at which $s_t = 0$ in a population of n individuals:

- Push: Pittel (1987) shows that $T(n) = \log n + \ln n + O(1)$
- Pull, Push-pull: Karp et al. (2000) shows that $T(n) = O(\log n) + O(\log \log n)$

Relevant bibliography:

- B. Pittel. On spreading a rumor. SIAM Journal of Applied Mathematics, 47(1):213–223, 1987.
- R. Karp, C. Schindelhauer, S. Shenker, B. Vocking. Randomized rumor spreading. In Proc. the 41st Symp. on Foundations of Computer Science, 2000.

Analytical results

In summary:

All methods converge to 0, but pull and push-pull are much more rapid

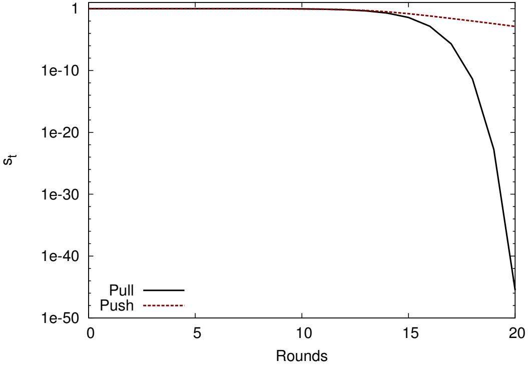


Figure: $n = 10\,000$

Comments

Benefits

- Simple epidemic eventually "infects" all the population
- It is extremely robust

Drawbacks

- Propagates updates much slower than direct mail (best effort)
- Requires examining contents of database even when most data agrees, so it cannot practically be used too often
- Normally used as support to best effort/rumor mongering, i.e. left running in the background

Working with multiple values

Examples of techniques to compare databases:

- Maintain checksum, compare databases if checksums unequal
- Maintain recent update lists for time T, exchange lists first
- Maintain inverted index of database by timestamp; exchange information in reverse timestamp order, incrementally re-compute checksums

Notes:

- Those ideas apply to databases
- Strongly application-dependent
- We will see how anti-entropy may be used beyond information dissemination

Rumor mongering in brief

- Nodes initially "susceptive"
- When a node receives a new update it becomes a "hot rumor" and the node "infective"
- A node that has a rumor periodically chooses randomly another node to spread the rumor
- Eventually, a node will "lose interest" in spreading the rumor and becomes "removed"
  - Spread too many times
  - Everybody knows the rumor
- A sender can hold (and transmit) a list of infective updates rather than just one

Rumor mongering: loss of interest

- **When**: Counter vs coin (random)
  - Coin (random): lose interest with probability $1/k$
  - Counter: lose interest after k contacts
- **Why**: Feedback vs blind
  - Feedback: lose interest only if the recipient knows the rumor.
  - Blind: lose interest regardless of the recipient.

Rumor mongering, Push

Blind/coin variant

```
upon update(v) do
    state ← INFECTED
    value ← v
    set timeout Δ

upon timeout do
    if state = INFECTED then
        q ← random(P)
        send ⟨PUSH, value⟩ to q
        if tossCoin(1/k) then
            state ← REMOVED
        set timeout Δ
```

Rumor mongering, Push

Blind/coin variant

upon receive $\langle \textsc{push}, v \rangle$ **do**
  if $state = \textsc{susceptible}$ **then**
    $value \leftarrow v$
    $state = \textsc{infected}$

Rumor mongering, Push

Feedback/counter variant

upon update(v) **do**
- $value \leftarrow v$
- $state \leftarrow$ INFECTED
- $counter \leftarrow k$
- **set timeout** Δ

upon timeout do
- **if** $state =$ INFECTED **then**
  - $q \leftarrow$ random(P)
  - **send** $\langle$PUSH$, p, value\rangle$ **to** q
  - **set timeout** Δ

Rumor mongering, Push

Feedback/counter variant

upon receive $\langle \text{PUSH}, q, v\rangle$ **do**
  send $\langle \text{REPLY}, \mathit{state}\rangle$ **to** q
  if $\mathit{state} = \text{SUSCEPTIBLE}$ **then**
    $\mathit{value} \leftarrow v$
    $\mathit{state} = \text{INFECTED}$
    $\mathit{counter} \leftarrow k$

upon receive $\langle \text{REPLY}, s\rangle$ **do**
  if $s \neq \text{SUSCEPTIBLE}$ **then**
    $\mathit{counter} \leftarrow \mathit{counter} - 1$
    if $\mathit{counter} = 0$ **then**
      $\mathit{state} \leftarrow \text{REMOVED}$

Analytical results

Question

How fast does the system converge to a state where all nodes are not infective? (inactive state)

Compartmental analysis again – Feedback, coin

Let s, i and r denote the fraction of susceptible, infective, and removed nodes, respectively. Then:

$$\begin{aligned} s + i + r &= 1 \\ ds/dt &= -si \\ di/dt &= +si - \frac{1}{k}(1-s)i \end{aligned}$$

Solving the differential equations:

- $s = e^{-(k+1)(1-s)}$
- Increasing k increases the probability that the nodes get the rumor

Quality measures

Residue

- The nodes that remain susceptible when the epidemic ends: value of s when $i = 0$.
- Residue must be as small as possible

Traffic

- The average number of database updates sent between nodes
- $m =$ total update traffic / # of nodes

Convergence

- t_{avg} : average time it takes for all nodes to get an update
- t_{max} : time it takes for the last node to get the update

Simulation results

Blind/coin				
Coin k	Residue s	Avg. Traffic m/n	Delay	
			t_{avg}	t_{max}
1	0.960	0.04	19.0	38.0
2	0.205	1.59	17.0	33.0
3	0.060	2.82	15.0	32.0
4	0.021	3.91	14.1	32.0
5	0.008	4.95	13.8	32.0

Simulation results

Feedback/counter				
Counter k	Residue s	Avg. Traffic m/n	Delay	
			t_{avg}	t_{max}
1	0.176	1.74	11.0	16.8
2	0.037	3.30	12.1	16.9
3	0.011	4.53	12.5	17.4
4	0.004	5.64	12.7	17.5
5	0.001	6.68	12.8	17.7

Push vs pull

- Push (what we have assumed so far)
  - If the database becomes quiescent, push stops trying to introduce updates
- Pull
  - If many independent updates, pull is more likely to find a source with a non-empty rumor list
  - But if the database is quiescent, several update requests are wasted

Rumor Mongering + Anti-Entropy

- Quick & Dirty Push Rumor Mongering
  - spreads updates fast with low traffic.
  - however, there is still a nonzero probability of nodes remaining susceptible after the epidemic
- Not-so-fast Push-Pull Anti-Entropy
  - can be run (infrequently) in the background to ensure all nodes eventually get the update with probability 1

Dealing with deletions

Deletion

- We cannot delete an entry just by removing it from a node - the absence of the entry is not propagated
- If the entry has been updated recently, there may still be an update traversing the network!

Death certificate

Replace the deleted item with a death certificate that has a timestamp and spreads like an ordinary update.

Dealing with deletions

Problem

We must, at some point, delete DCs or they may consume significant space

- **Strategy 1**: retain each DC until all nodes have received it
  - requires a protocol to determine which nodes have it and to handle node failures
- **Strategy 2**: hold DCs for some time (e.g. 30d) and discard them
  - pragmatic approach
  - we still have the "resurrection" problem
  - increasing the time requires more space

Dormant certificates

Observation:

- we can delete very old DCs but retain only a few "dormant" copies in some nodes
- if an obsolete update reaches a dormant DC, it is "awakened" and re-propagated

Analogy with epidemiology:

- the awakened DC is like an antibody triggered by an immune reaction

Epidemics vs probabilistic broadcast

Anti-entropy:
DB is a collection of message received

Rumor mongering
Updates $\equiv$ Message broadcast

Table of contents

What's next?

Problems

- Membership:
  How do processes get to know each other, and how many do they need to know?
- Network awareness:
  How to make the connections between processes reflect the actual network topology such that the performance is acceptable?
- Buffer management:
  Which updates to drop when the storage buffer of a process is full?

Does not end here...

Epidemic/gossip approach has been successfully applied to other problems. We will provide a brief overview of them.

Reading Material

- A. J. Demers, D. H. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. E. Sturgis, D. C. Swinehart, and D. B. Terry. Epidemic algorithms for replicated database maintenance.

  In *Proc. of the 6th annual ACM Symposium on Principles of Distributed Computing Systems (PODC'87)*, pages 1–12, 1987.

  http://www.disi.unitn.it/~montreso/ds/papers/demers87.pdf

- A. Montresor. Gossip and epidemic protocols.

  Wiley Encyclopedia of Electrical and Electronics Engineering, 2017.

  http://www.disi.unitn.it/~montreso/ds/papers/montresor17.pdf